Nous allons, dans cette première partie, traiter les **concepts de base du PHP Orienté Objet**.

Qu'est-ce qu'un objet ?

Dans tous les langages de programmation, nous pouvons stocker des valeurs dans des emplacements de la mémoire de l'ordinateur. Dans certains langages on appellera ces emplacements des registres, dans d'autres des variables.

Les variables en PHP peuvent être de plusieurs types :

- Numérique (entier, décimal)
- Chaîne de caractères
- Tableaux

Ces types de variables sont assez statiques dans leur utilisation, nous pouvons stocker et lire la ou les valeurs qu'elles contiennent, mais pas plus.

Les **objets** pourraient être comparés à des **variables** dans lesquelles on pourra non seulement **écrire ou lire des valeurs**, mais également **effectuer des actions**.

Exemples de la vie quotidienne

Dans notre vie quotidienne, **nous utilisons des objets tous les jours**. Notre cafetière, notre voiture, notre téléviseur sont autant d'objets "quotidiens" qui pourraient se comparer à un objet PHP.

En effet, chacun de ces objets aura des **caractéristiques** qui lui sont propres, sa couleur, son énergie, sa taille, le nombre de places de la voiture. Chacune de ces caractéristiques vont définir notre objet.

Nous pourrons également effectuer des **actions** avec chacun de ces objets, démarrer, allumer, changer de chaine, accélérer...

Et le PHP dans tout ça ?

En PHP, les **caractéristiques** s'appelleront des **propriétés** et les **actions** s'appelleront des **méthodes**. Nos objets pourront posséder ces propriétés et ces méthodes en fonction de ce que nous souhaitons.

Par exemple, nous pourrons créer un **objet** pour stocker un **compte bancaire**.

Un compte bancaire aura des **propriétés** qui lui seront propres, un **titulaire**, un **solde**, un **taux d'intérêts**...

Il aura également des **méthodes** qui permettront de **déposer** de l'argent, d'en **retirer**, de **consulter** le solde...

Conventions

Avant d'aller plus loin, voici quelques **conventions importantes** dans le développement orienté objet.

- Chaque objet aura son propre fichier
- Le nom de fichier sera le nom de l'objet
- Le nom de l'objet sera écrit en UpperCamelCase

Premier objet

Un objet en PHP se définit par le mot clé "**class**". Nous allons donc créer l'objet "**Compte**" dans un fichier intitulé "**Compte.php**" (notez le C majuscule). Nous placerons ce fichier dans un dossier "**classes**" qui contiendra tous nos fichiers de classes.

Le fichier "**Compte.php**" contiendra donc la déclaration de classe "**Compte**" comme ceci

```
class Compte
{


}
```

Une fois la classe déclarée, à l'intérieur nous allons pouvoir déclarer les **propriétés de l'objet**. Notre compte pourra avoir un **titulaire** et un **solde**. Nous allons donc les déclarer comme ci-dessous (nous reviendrons sur le mot clé "public" dans la partie "Visibilité").

```
class Compte
{
    public $titulaire;
    public $solde;
}
```

Comme nos objets pourront devenir rapidement complexes, je vous conseille de les **commenter** dès le départ.

Le même code commenté sera le suivant

```php
/**
 * Classe correspondant à un compte bancaire
 */
class Compte
{
    /**
     * Titulaire du compte
     * @var string
     */
    public $titulaire;

    /**
     * Solde du compte
     * @var float
     */
    public $solde;
}
```

Instancier l'objet

Après avoir déclaré la classe, nous pourrons l'appeler à plusieurs reprises dans des variables différentes. On appellera cette manipulation "**Instancier l'objet**" qui permet de créer plusieurs exemplaires de notre objet, chaque exemplaire étant appelé "**instance**".

Nous allons créer un fichier "**index.php**" dans lequel nous allons instancier notre objet à 2 reprises.

Pour commencer, nous devons appeler le fichier "**Compte.php**" au moyen d'un "**require_once**" comme ceci

```
require_once
'classes/Compte.php';
```

Une fois le fichier inclu, nous pouvons l'instancier à 2 reprises, par exemple, comme ceci

```
// Première instance de la
classe Compte
$compte1 = new Compte;

// Deuxième instance de la
classe Compte
$compte2 = new Compte;
```

Si nous faisons un "**var_dump**" de la variable "**$compte1**", nous aurons le résultat suivant

```
object(Compte)[1]
  public 'titulaire' => null
  public 'solde' => null
```

Nous remarquons que **les propriétés de notre objet ne sont pas remplies**.

Modifier la valeur des propriétés

Lorsque nous avons instancié l'objet "**Compte**", nous n'avons **pas défini de valeurs pour les propriétés "titulaire" et "solde"**.

Nous allons pouvoir les modifier comme ceci

```php
// On attribue un titulaire
$compte1->titulaire =
'Benoit';

// On attribue un solde
 $compte1->solde = 500;
```

Le "**var_dump**" affiche maintenant les valeurs suivantes

```
object(Compte)[1]
  public 'titulaire' => string 'Benoit' (length=6)
  public 'solde' => int 500
```

Le constructeur

Comme nous l'avons vu précédemment, **lorsque nous instancions notre objet, ses propriétés ne sont pas remplies**.

Nous allons pouvoir utiliser une méthode (fonction) spécifique appelée "**constructeur**" qui va nous permettre de **"construire" notre objet lorsque nous l'instancions**.

Cette méthode sera **exécutée automatiquement lors de l'instanciation** de l'objet.

Retournons dans notre classe "**Compte**" et ajoutons la méthode ci-dessous juste après la propriété "**solde**"

```
/**
 * Constructeur de notre objet Compte
 * @param string $titulaire Titulaire du compte
 * @param float $solde Solde du compte
 */
public function __construct(string $titulaire, float
$solde)
{
    // On affecte le titulaire à la propriété titulaire
    $this->titulaire = $titulaire;

    // On affecte le solde à la propriété solde
    $this->solde = $solde;
}
```

Vous remarquerez l'utilisation de "**this**" qui représente l'instance de classe que nous utilisons.

Pour instancier notre classe, nous devrons donc maintenant **préciser un titulaire et un solde** de cette façon

```php
// Première instance de la classe Compte
 $compte1 = new Compte('Benoit', 500);
```

Nous pouvons également **définir des valeurs par défaut** dans notre constructeur en y ajoutant une valeur comme ceci (pour le solde)

```php
/**
 * Constructeur de notre objet Compte
 * @param string $titulaire Titulaire du compte
 * @param float $solde Solde du compte
 */
public function __construct(string $titulaire, float $solde = 500)
{
    // On affecte le titulaire à la propriété titulaire
    $this->titulaire = $titulaire;

    // On affecte le solde à la propriété solde
    $this->solde = $solde;
}
```

Avec cette modification nous pourrons instancier la classe de 2 façons

```php
// Première façon d'instancier la classe Compte
$compte1 = new Compte('Benoit', 2000);

// Deuxième façon d'instancier la classe Compte
  $compte1 = new Compte('Benoit');
```

Ajouter des méthodes

Une fois notre objet créé, nous pouvons avoir besoin d'**effectuer des actions** sur celui-ci.

Dans le cas de notre "**Compte**", nous devrions pouvoir **afficher le solde**, **déposer** et **retirer** de l'argent.

Pour **afficher le solde du compte**, nous allons créer une méthode que nous placerons directement sous le constructeur.

Notre méthode affichera le solde du compte sous la forme d'un "**echo**" et sera créée comme ceci

```php
/**
 * Voir le solde du compte
 * @return void
 */
public function voirSolde()
{
    echo "Le solde du compte est de $this->solde euros";
}
```

Pour afficher le solde de notre "**compte1**", nous allons donc utiliser la méthode de la façon suivante

```php
$compte1->voirSolde();
```

Pour pouvoir ajouter de l'argent à notre solde, nous allons créer une méthode "**deposer**"

```php
/**
 * Déposer de l'argent sur le compte
 *
 * @param float $montant Montant déposé
 * @return void
 */
public function deposer(float $montant)
{
    // On vérifie si le montant est positif
    if($montant > 0){
        $this->solde += $montant;
    }
}
```

Et l'utiliser comme ceci

```php
$compte1->deposer(1000);
```

Pour le retrait, nous allons créer une méthode "**retirer**" qui vérifiera le solde avant d'autoriser le retrait

```php
/**
 * Retire un montant du solde du compte
 *
 * @param float $montant Montant à retirer
 * @return void
 */
public function retirer(float $montant)
{
    // On vérifie le montant et le solde
    if($montant > 0 && $this->solde >= $montant){
        $this->solde -= $montant;
    }else{
        echo "Montant invalide ou solde insuffisant";
    }
}
```

Et l'utiliser comme ceci

```php
$compte1->retirer(500);
```